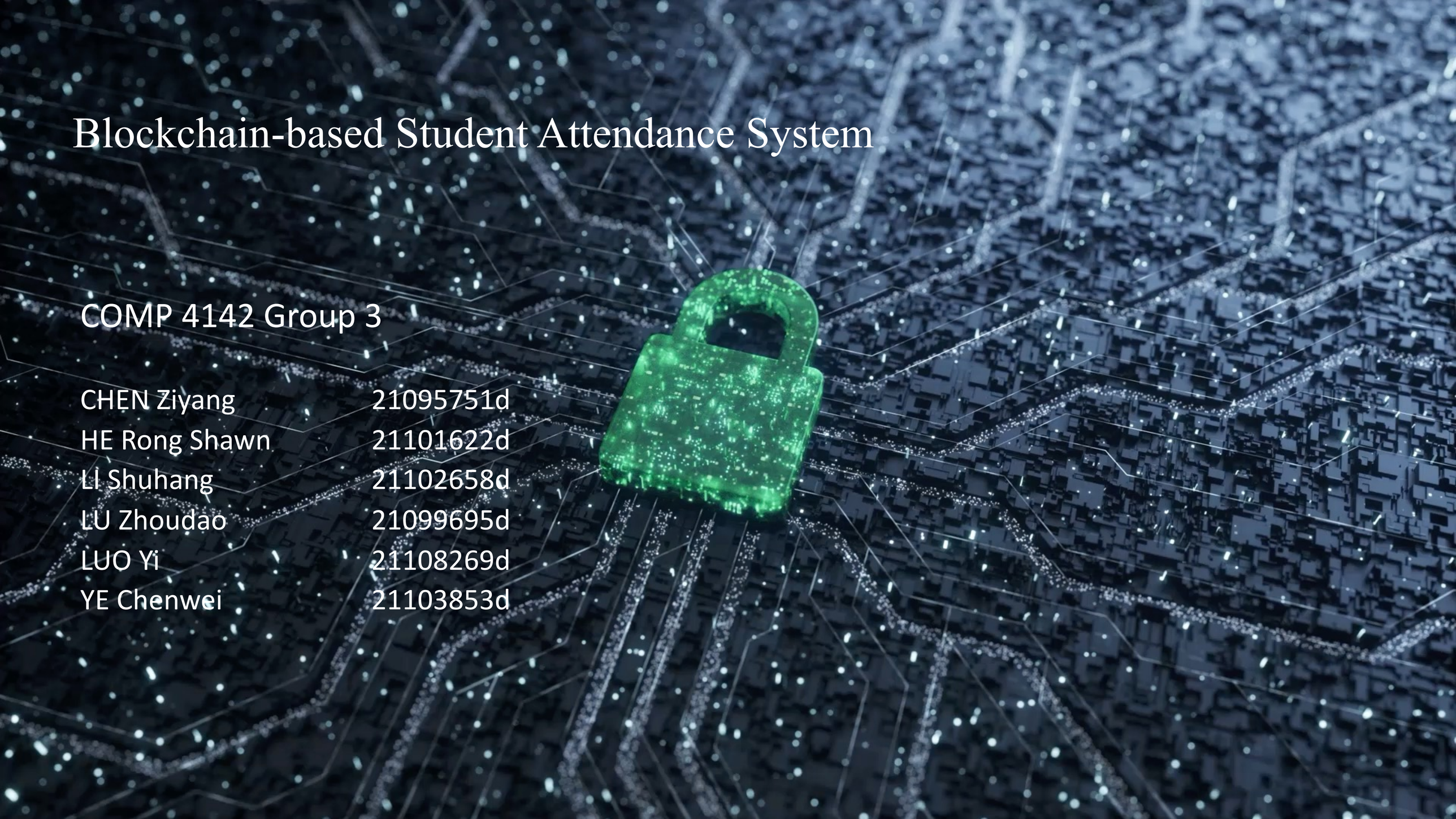
Blockchain-based Student Attendance System



COMP 4142 Group 3

CHEN Ziyang	21095751d
HE Rong Shawn	21101622d
LI Shuhang	21102658d
LU Zhoudao	21099695d
LUO Yi	21108269d
YE Chenwei	21103853d

Contributors

Student Name	Student ID	Responsible Part
CHEN Ziyang	21095751d	All frontend content of the Student Attendance Application, and create a user-friendly UI. Create the function "Get Secret Key" on the backend. Assist the backend to complete the work.
HE Rong	21101622d	Assist in enhancing functionality and organizing meetings. Overall flow check.
LI Shuhang	21102658d	Implementation the backend of the Student Attendance Application: Student information registration and Mint. Assist the frontend to complete the corresponding work.
LU Zhoudao	21099695d	Enhanced functionality: Achieve dynamic difficulty and Basic Fork Resolution (cumulative difficulty).
LUO Yi	21108269d	Enhanced functionality: Basic Fork Resolution.
YE Chenwei	21103853d	Implementation the backend of the Student Attendance Application: Attendance information recording and Record querying. Assist the frontend to complete the corresponding work.

Content

System Architecture Overview

Brief introduce

Attendance Record

Register user (teacher & student)
Mint unconfirmed transaction
Public event
List event ID
Record attendance
Query

Enhanced functionality implementation

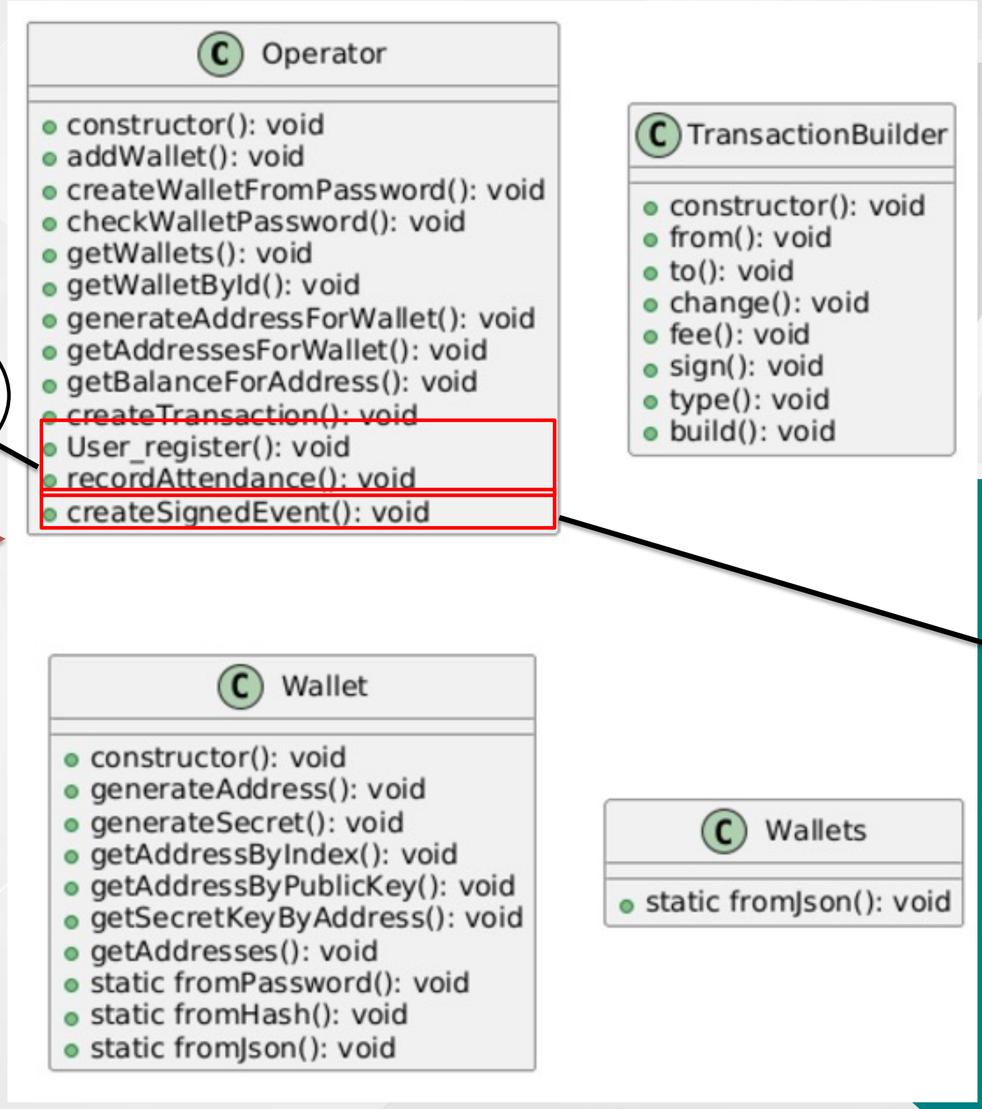
Achieve dynamic difficulty
Basic Fork Resolution

System Architecture Overview

- blockchain
- httpServer
- miner
- node
- operator
- util
- .DS_Store
- config.js
- naivecoin.js

registration

Attendance information recording



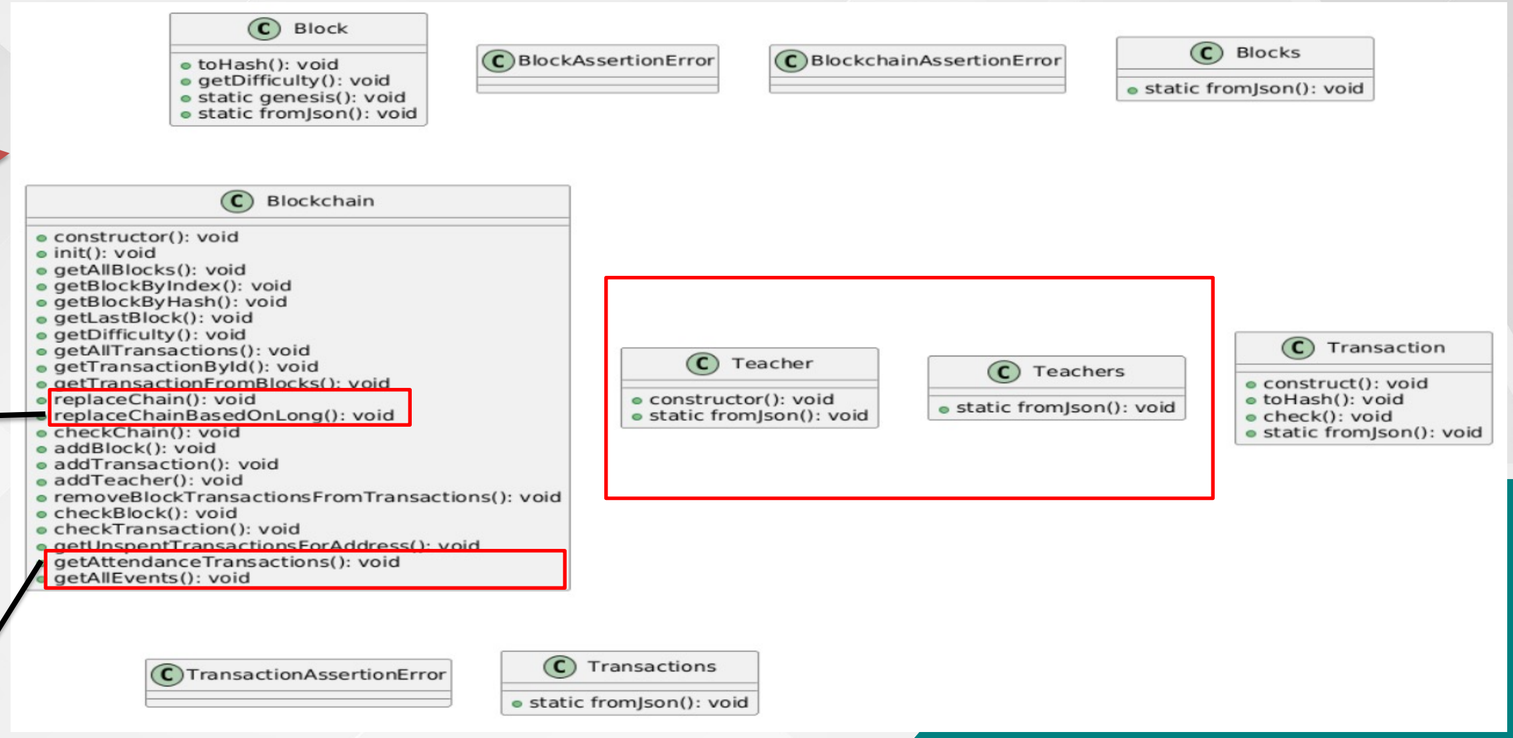
Teachers can use their private keys to create events and set deadlines

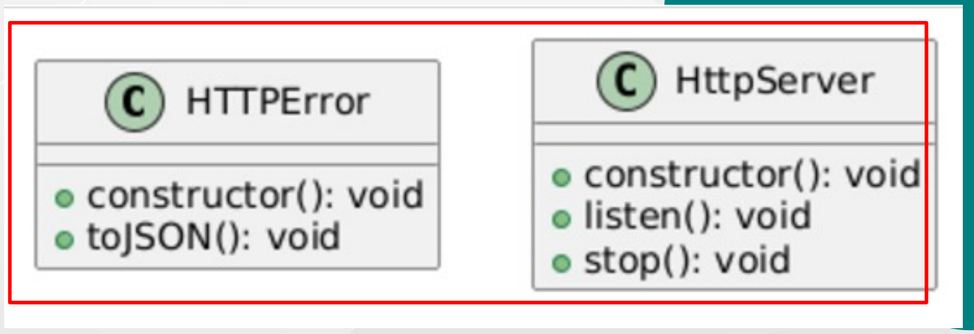
System Architecture Overview

- blockchain
- httpServer
- miner
- node
- operator
- util
- .DS_Store
- config.js
- naivecoin.js

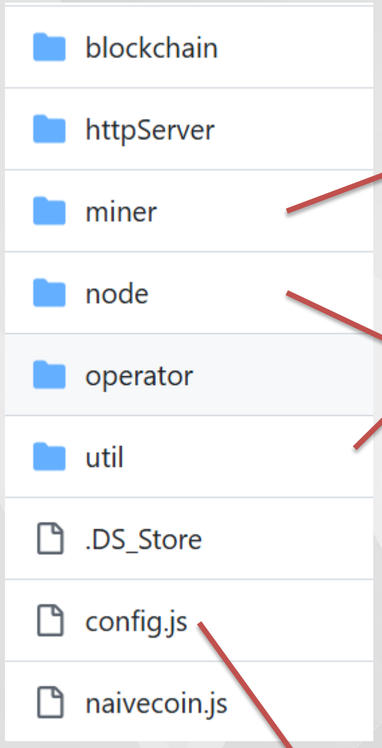
Fork Resolution

Record querying



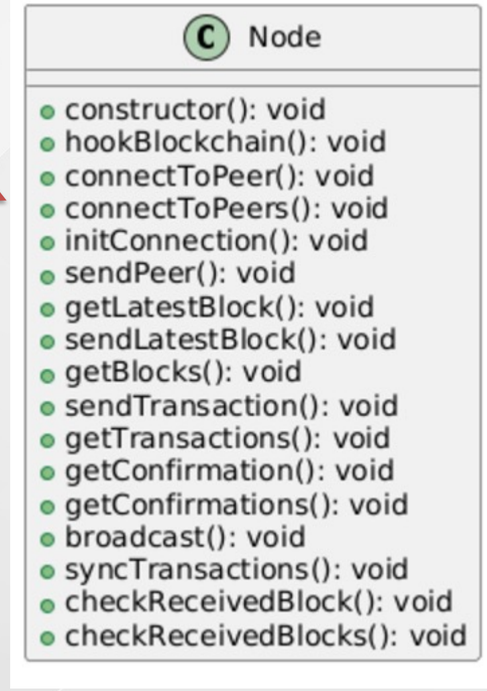


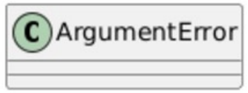
System Architecture Overview

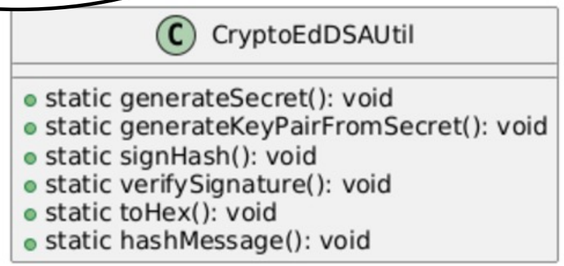


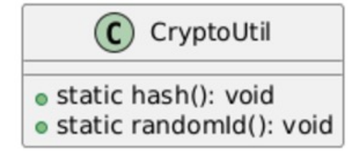
Dynamic difficulty

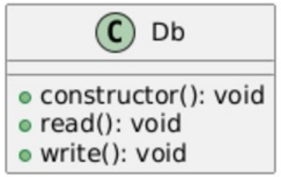
Mint, check correctness

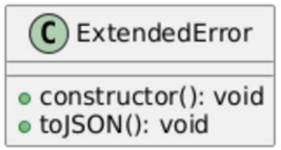












Function Component:
User Registration

1. **Students**
2. **Teachers**

```
async User_register(User_ID, password) {
    const newWallet = this.createWalletFromPassword(password);
    const publicKeyAddress = this.generateAddressForWallet(newWallet.id);
    const registrationTransaction = {
        id: `reg_${User_ID}`,
        type: 'registration',
        data: {
            inputs: [],
            outputs: [
                {
                    User_ID: User_ID,
                    publicKey: publicKeyAddress,
                    amount: 0,
                    address: NaN
                }
            ]
        }
    };

    await this.blockchain.addTransaction(registrationTransaction);
    console.log("newWallet:", newWallet);
    console.log("publicKeyAddress:", publicKeyAddress);
    console.log(`Student ${User_ID} registered with wallet ID: ${newWallet.id} and public address: ${publicKeyAddress}`);

    return {
        User_ID: User_ID,
        walletID: newWallet.id,
        publicKey: publicKeyAddress
    };
}
```

Function Component:
Publish Event ID

1. Verify user identity
2. Create the DDL
3. Construct transaction

```
async createSignedEvent(User_ID, eventID, privateKey, ddl) {
    console.log("Teachers array:", this.blockchain.teachers);
    const isTeacher = this.blockchain.teachers.some(teacher => teacher.id === User_ID);
    console.log(isTeacher);
    if (!isTeacher) {
        throw new Error(`User ${User_ID} is not a teacher and cannot create events.`);
    }
    const parsedDate = new Date(ddl);
    const timestamp = parsedDate.getTime();
    const messageHash = CryptoEdDSAUtil.hashMessage(`${eventID}_${timestamp}`);
    const signature = CryptoEdDSAUtil.signHash(privateKey, messageHash);

    const eventTransaction = {
        id: `event_${eventID}`,
        type: 'event',
        data: {
            inputs: [],
            outputs: [
                {
                    User_ID: User_ID,
                    eventID: eventID,
                    timestamp: timestamp,
                    signature: signature,
                    amount: 0,
                    address: NaN
                }
            ]
        }
    };

    await this.blockchain.addTransaction(eventTransaction);
    console.log(`Event created and signed by teacher ${User_ID} with ddl: ${ddl}`);
    return { transactionId: eventTransaction.id, timestamp };
}
```

Function Component:
Mint Unconfirmed Transaction

1. For 'event' and 'attendance'

2. For other types

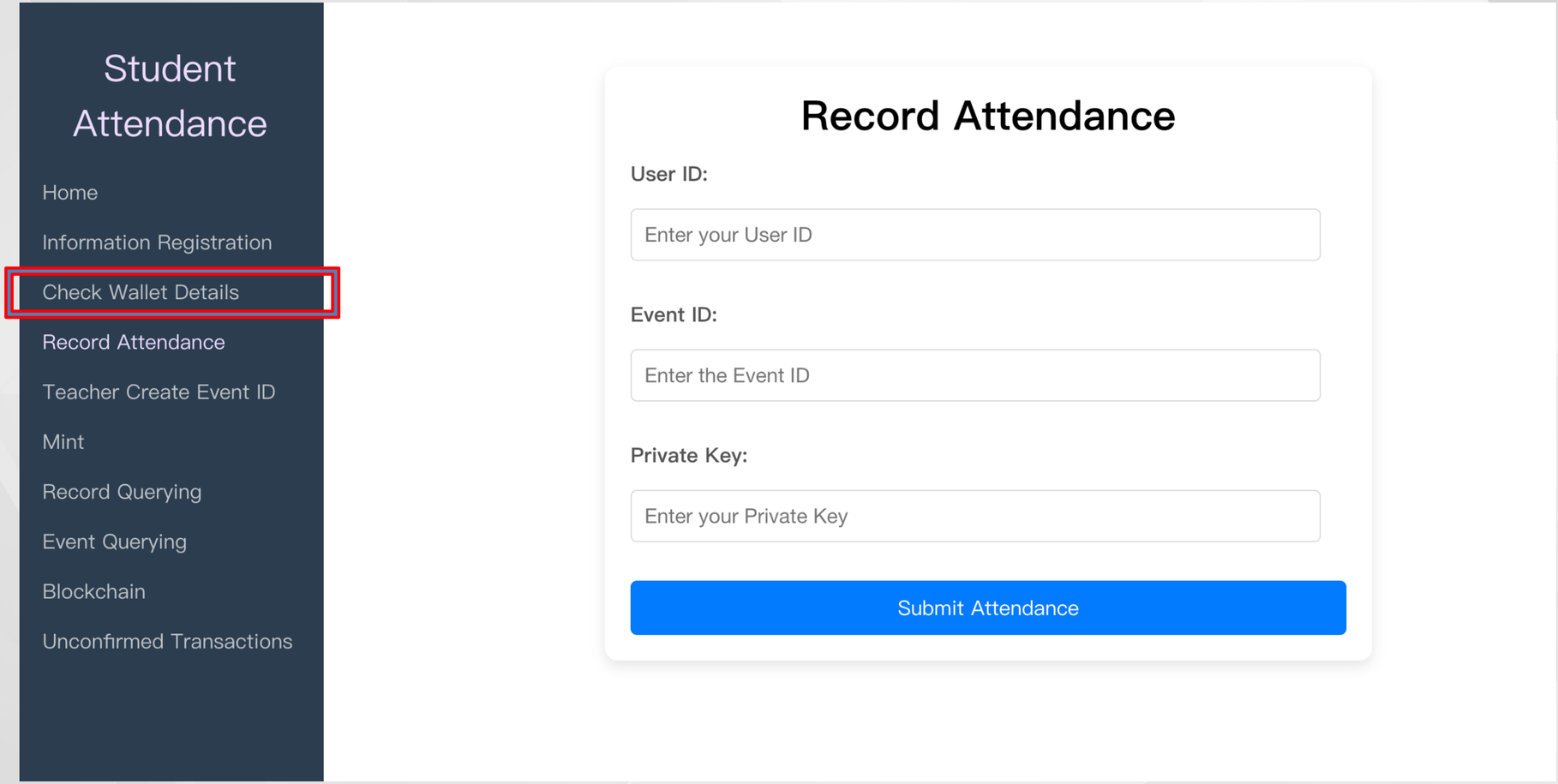
```
if (R.all(R.equals(false), transactionInputFoundAnywhere)) {
    if (transaction.type === 'event' || transaction.type === 'attendance') {
        const isValid = Miner.verifyEventTransaction(transaction, blockchain);
        console.log(isValid);
        if (isValid) {
            selectedTransactions.push(transaction);
        } else {
            rejectedTransactions.push(transaction);
        }
    } else if (transaction.type === 'registration') {
        selectedTransactions.push(transaction);
    } else if (transaction.type === 'regular' && negativeOutputsFound === 0) {
        selectedTransactions.push(transaction);
    } else if (transaction.type === 'reward') {
        selectedTransactions.push(transaction);
    } else if (negativeOutputsFound > 0) {
        rejectedTransactions.push(transaction);
    }
} else {
    rejectedTransactions.push(transaction);
}
candidateTransactions);
```

Function Component:
Mint Unconfirmed Transaction

```
static verifyTransaction(transaction, blockchain) {
    const currentTime = parseInt(new Date().getTime(),10);
    if (!transaction.data.outputs || transaction.data.outputs.length === 0) {
        console.warn(`Transaction ${transaction.id} is missing outputs.`);
        return false;
    }

    const userid = transaction.data.outputs[0].User_ID;

    //console.log(currentTime);
    //console.log(transaction.data.outputs[0].timestamp);
    //console.log(parseInt(transaction.data.outputs[0].timestamp,10));
    let event_ddl = null;
    const blocks = blockchain.getAllBlocks();
    if (transaction.type === 'attendance'){
        const eventid = transaction.data.outputs[0].eventID;
        for (const block of blocks) {
            for (const t of block.transactions) {
                if (t.type === 'event') {
                    //console.log(t.output.timestamp);
                    const output = t.data.outputs.find(output => output.eventID === eventid);
                    if (output) {
                        event_ddl = output.timestamp;
                        break;
                    }
                }
            }
            if (event_ddl) break;
        }
        console.error(event_ddl);
        console.error(transaction.data.outputs[0].timestamp);
        if (parseInt(event_ddl,10) < parseInt(transaction.data.outputs[0].timestamp,10)){
            console.warn(`Transaction ${transaction.id} rejected: expired.`);
            console.error(event_ddl);
        return false;
        }
        return true;
    }
```

```
if (currentTime > parseInt(transaction.data.outputs[0].timestamp,10)) {
    console.warn(`Transaction ${transaction.id} rejected: expired.`);
    return false;
}
const eventHash = CryptoEdDSAUtil.hashMessage(
    `${transaction.data.outputs[0].eventID}_${transaction.data.outputs[0].timestamp}`
);
//const userid = transaction.data.outputs[0].User_ID;
let userPublicKey = null;
for (const block of blocks) {
    for (const t of block.transactions) {
        if (t.type === 'registration') {
            const output = t.data.outputs.find(output => output.User_ID === userid);
            if (output) {
                userPublicKey = output.publicKey;
                break;
            }
        }
    }
    if (userPublicKey) break;
}
console.log(userPublicKey);
const isSignatureValid = CryptoEdDSAUtil.verifySignature(
    userPublicKey,
    transaction.data.outputs[0].signature,
    eventHash
);
console.log(transaction.data.outputs[0].signature);

if (!isSignatureValid) {
    console.warn(`Transaction ${transaction.id} rejected: signature verification failed.`);
    return false;
}
return true;
```

Function Component:
Record

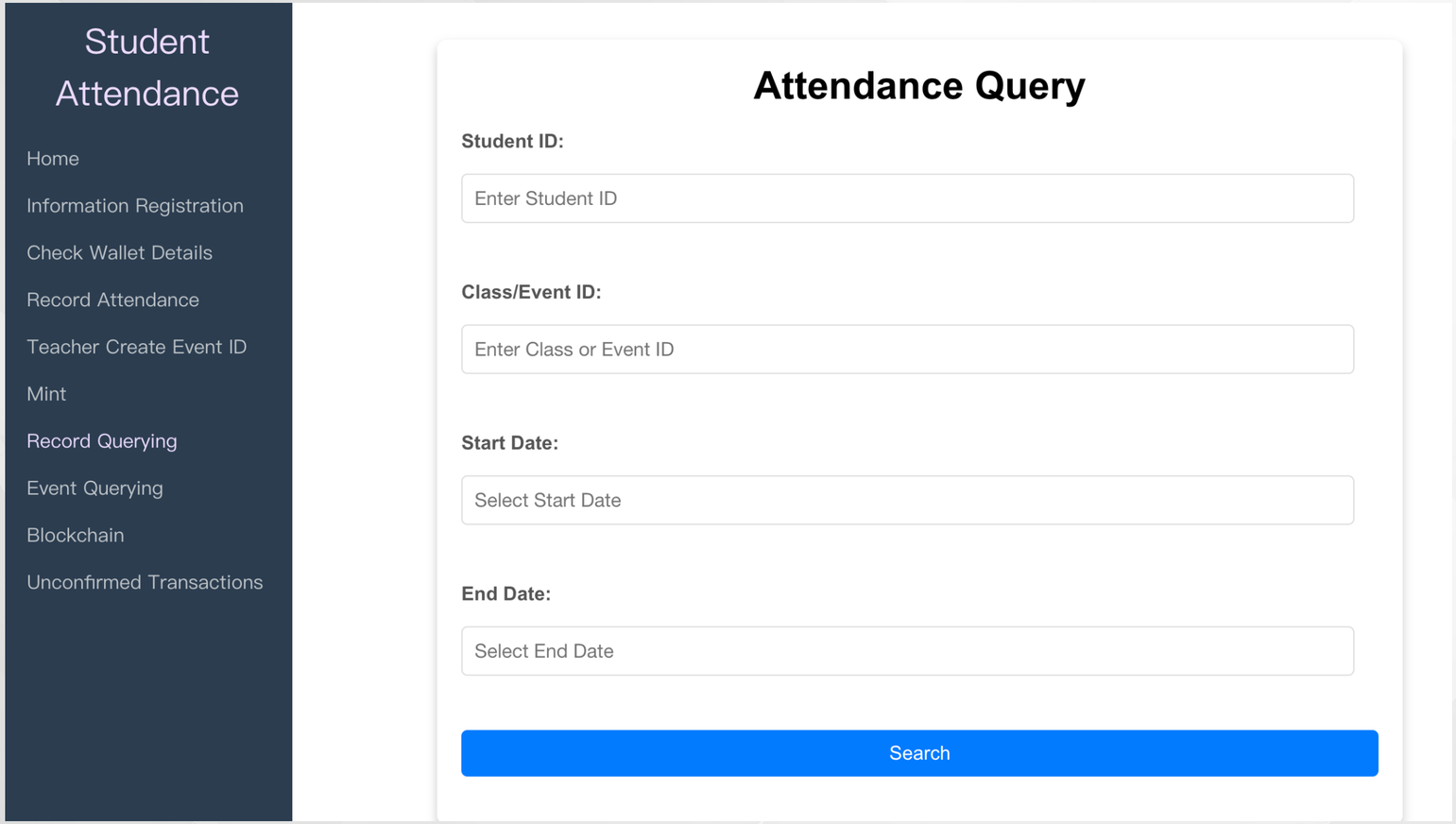


Function Component:

Record

```
async recordAttendance(User_ID, eventID, timestamp, signature) {
    const attendanceTransaction = {
        id: `${User_ID}_${eventID}_${timestamp}`,
        type: 'attendance',
        data: {
            inputs: [],
            outputs: [
                {
                    User_ID: User_ID,
                    eventID: eventID,
                    timestamp: timestamp,
                    signature: signature,
                    amount: 0,
                    address: NaN
                }
            ]
        }
    };

    try {
        await this.blockchain.addTransaction(attendanceTransaction);
        console.log(`Attendance recorded for student ID: ${User_ID}, event ID: ${eventID} at timestamp: ${timestamp}`);

        return {
            message: "Attendance recorded successfully",
            transactionId: attendanceTransaction.id
        };
    } catch (error) {
        console.error("Failed to record attendance:", error.message);
        throw new Error("Failed to record attendance");
    }
}
```

Function Component:
Query Search



1. By User ID
2. By Event ID
3. By Starting Time
4. By Ending Time

Function Component:
Query Search

```
getAttendanceTransactions({ User_ID, eventID, startTime, endTime }) {
    const transactionsInBlocks = R.flatten(R.map(R.prop('transactions'), this.getAllBlocks()));
    const parsedStartTime = startTime ? new Date(startTime).getTime() : null;
    const parsedEndTime = endTime ? new Date(endTime).getTime() : null;

    return transactionsInBlocks.filter(tx => {
        if (tx.type !== 'attendance') return false;

        const isStudentMatch = !User_ID || R.propEq('User_ID', User_ID, tx.data);
        const isEventMatch = !eventID || R.propEq('eventID', eventID, tx.data);
        const isTimeMatch = (!parsedStartTime || tx.data.timestamp >= parsedStartTime) &&
                            (!parsedEndTime || tx.data.timestamp <= parsedEndTime);

        return isStudentMatch && isEventMatch && isTimeMatch;
    });
}
```

Enhanced functionality implementation:

Achieve dynamic difficulty

In bitcoin:

- BLOCK_GENERATION_INTERVAL=10 minute
- DIFFICULTY_ADJUSTMENT_INTERVAL1=2016 blocks

```
const EVERY_X_BLOCKS = 10;
const TARGET_TIME = 10;
```

Increase the difficulty when this time is less than the expected time.

Decrease the difficulty when this time is bigger than the expected time.

```
newDifficulty = currentDifficulty * (TARGET_TIME / averageTime);
```

Enhanced functionality implementation:

Basic Fork Resolution

```
async replaceChainBasedOnLong(newBlockchain) {
    //        It doesn't make sense to replace this blockchain by a smaller one
    if (newBlockchain.length <= this.blocks.length) {
        console.error('Blockchain shorter than the current blockchain');
        throw new BlockchainAssertionError('Blockchain shorter than the current blockchain');
    }

    // Verify if the new blockchain is correct
    await this.checkChain(newBlockchain);
}
```

```
async replaceChain(newBlockchain) {
     // Calculate cumulative difficulties
    const currentCumulativeDifficulty = this.calculateCumulativeDifficulty(this.blocks);
    const newCumulativeDifficulty = this.calculateCumulativeDifficulty(newBlockchain);

    // Compare cumulative difficulties
    if (newCumulativeDifficulty > currentCumulativeDifficulty) {
        console.info('Replacing current blockchain with received blockchain due to higher cumulative difficulty');
        let newBlocks = R.takeLast(newBlockchain.length - this.blocks.length, newBlockchain);

        R.forEach((block) => {
            this.addBlock(block, false);
        }, newBlocks);

        this.emitter.emit('blockchainReplaced', newBlocks);
    } else {
        console.warn('Received chain has lower cumulative difficulty. Ignoring it.');
    }

    // Get the blocks that diverges from our blockchain
    console.info('Received blockchain is valid. Replacing current blockchain with received blockchain');
    let newBlocks = R.takeLast(newBlockchain.length - this.blocks.length, newBlockchain);
}
```

```
calculateCumulativeDifficulty(blocks) {
    return R.sum(R.map(block => Math.pow(2, block.getDifficulty()), blocks));
}

syncWithPeers(peerChains) {
    try {
        for (const newChain of peerChains) {
            this.replaceChain(newChain);
        }
    } catch (err) {
        console.warn('Fork resolution failed for some chains:', err.message);
    }
}
```

Front-end: React Framework

The front-end is built with **React** and **React Router** for efficient navigation and modular code. Components like **Layout** and **Sidebar** ensure a clear, user-friendly interface and seamless interaction.

- Initialize a React project:

```
MacBook-Air naivecoin % npx create-react-app client
```

- Install necessary dependencies:

```
MacBook-Air naivecoin % cd client
MacBook-Air client % npm install axios react-router-dom react-datepicker
```

- To allow the frontend (usually running in a browser) to make cross-origin requests to the backend server, and to resolve the browser's same-origin policy restrictions, we use CORS (Cross-origin Resource Sharing)

```
const cors = require('cors');
```

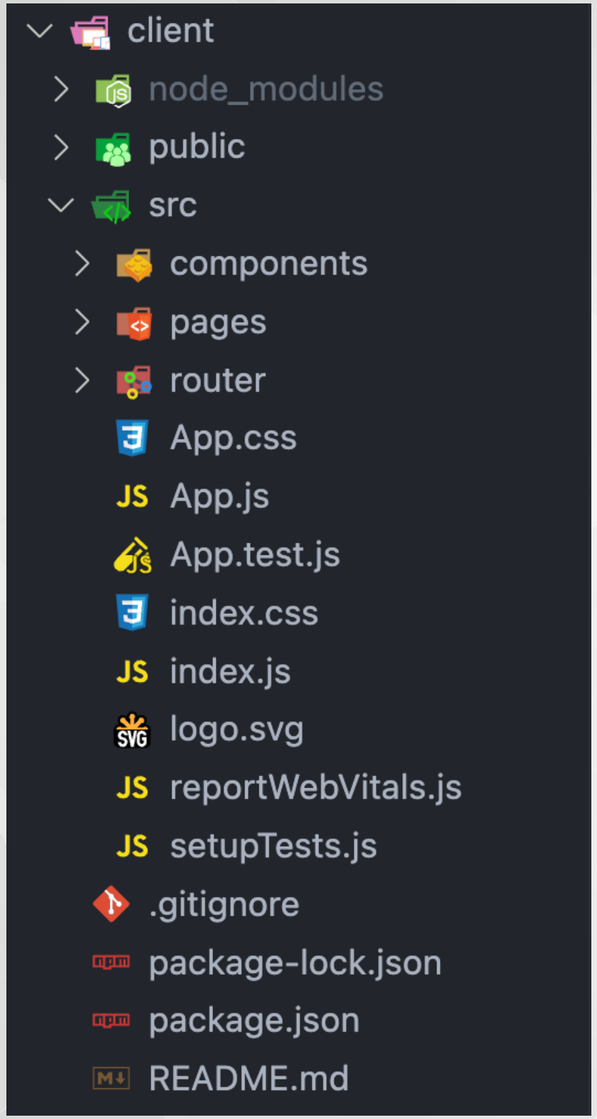
```
this.app = express();
this.app.use(cors({
    origin: 'http://localhost:3000',
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    credentials: true,
    allowedHeaders: 'Content-Type, Authorization'
}));
```

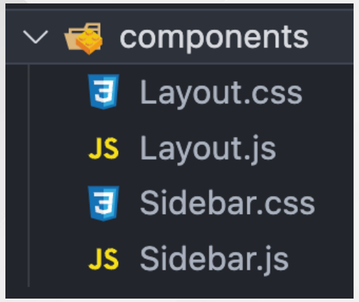
- Run the React development server:

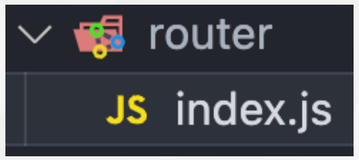
```
MacBook-Air naivecoin % cd client
MacBook-Air client % npm start
```

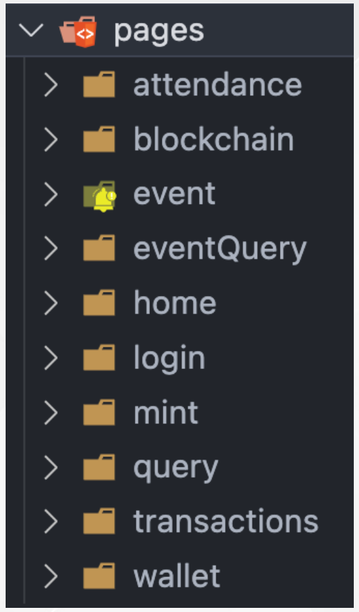
- The development server runs at http://localhost:3000.

Front-end: React Framework










```
// src/App.js

import React from 'react';
import { RouterProvider } from 'react-router-dom';
import router from './router';

function App() {
  return <RouterProvider router={router} />;
}

export default App;
```

- **components**: Includes reusable UI components: Layout.js and Sidebar.js.
- **pages**: Contains different pages or views of the application, such as attendance, event, query, etc.
- **router**: Contains routing configuration, setting up routes for the app.

- **The App.js file** sets up React Router using RouterProvider and relies on a router.js file to manage routes and handle navigation.

Demonstration

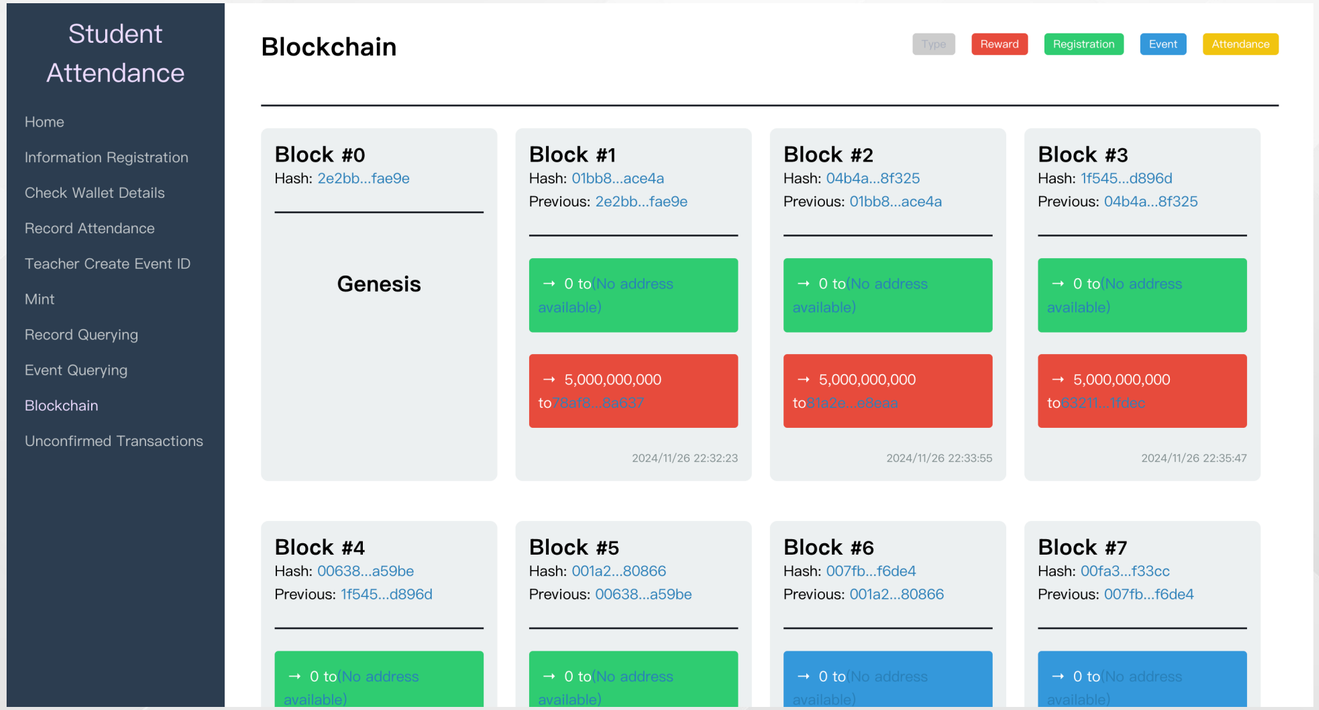
- Run a node:

```
-MacBook-Air naivecoin % node bin/naivecoin.js -p 3001 --name 1
```

- Run the React development server:

```
-MacBook-Air naivecoin % cd client
-MacBook-Air client % npm start
```

- The development server runs at http://localhost:3000.



Thank You

For Your Attention